

Ejercicio Práctico

 **Tema:** Seguridad en APIs RESTful y mitigación de vulnerabilidades OWASP

Objetivo del ejercicio:

Analizar una API vulnerable, detectar fallos relacionados con el OWASP Top 10, y aplicar buenas prácticas para mitigar riesgos como **Broken Access Control**, **Inyección**, **XSS** y **Falta de autenticación robusta**.

Escenario:


Se te entrega un fragmento de una API en PHP que expone un endpoint para eliminar usuarios:

```
<?php
$id = $_GET['id'];
if ($_GET['role'] === 'admin') {
    $conexion->query("DELETE FROM usuarios WHERE id = $id");
    echo "Usuario eliminado";
} else {
    echo "Acceso denegado";
}
?>
```



Parte 1: Análisis

a) ¿Cuáles son las vulnerabilidades presentes en este código?
(Selecciona todas las que correspondan)

- Inyección SQL 
- Broken Access Control 

- Cifrado débil
 - Falta de autenticación/autorización robusta 
-

b) ¿Qué podría hacer un atacante si conoce este endpoint?

- Cambiar el fondo del sitio
 - Eliminar usuarios sin estar autenticado 
 - Manipular el `id` en la URL para borrar usuarios arbitrarios 
-

Parte 2: Reescritura segura del código

Instrucciones:

Reescribe el código cumpliendo las siguientes medidas de seguridad:

1. Reemplaza `$_GET['role']` con verificación por **JWT o sesión autenticada**.
 2. Usa consultas preparadas para eliminar el usuario.
 3. Verifica que el usuario tenga el rol correcto desde una fuente segura (por ejemplo, la sesión).
-

Parte 3: Aplicación práctica (extra)

Opcional:

Simula este endpoint usando Postman o curl, primero de forma insegura y luego de forma segura. Compara los resultados cuando:

- No hay sesión activa
- El rol no es admin
- El `id` es manipulado desde el cliente

Reflexión final:

- ¿Qué ventaja tiene usar roles desde sesión en vez de parámetros GET?
 - ¿Por qué es peligroso construir consultas directamente con valores de entrada?
-